Realtime publishers

The Essentials Series: Mainframe Application Modernization

Is Your Mainframe Application Working as Hard as You Are?

sponsored by

by Don Jones

Copyright Statement

Is Your Mainframe Application Working as Hard as You Are?

Many of today's most successful companies run critical portions of their IT operations on a mainframe or midrange computer—and have done so for years and years. Mainframe applications offer power, stability, and reach—all things that enterprises rely on. In recent years, mainframe vendors have kept up with the times, offering the ability to host Web-based applications, email and collaboration software, and much more.

But mainframe applications present a problem, too: They lock up all your most important data in what is, in many ways, a black box. Mainframe applications tend to be highly proprietary and often include much of their business logic within the presentation layer, making it difficult, impractical, or outright impossible to safely access the application data by any other means.

But today's businesses *need* to utilize their mainframe applications in *many* other ways. Today's businesses need to move faster, be more flexible, and offer more services to internal and external customers. How can they do so when their application is locked up in a box?

This *Essential Series* guide will explore exactly that problem: How to bring your mainframe applications into the modern world, how to leverage mainframe-based data in powerful and exciting new ways, and—most importantly—how to do so without losing or compromising your existing mainframe investment.

Setting the Scene

Before we dive in, let's take a moment to make sure you and I are on the same page with what constitutes a mainframe application. I think this is important only because many folks are using their mainframes in creative new ways, and some of those newer and more flexible ways don't have the same problems as a more traditional mainframe application.

Traditional mainframe applications—the kind I'll be discussing in this guide—are commonly used for bulk data processing, such as census data. They're also used to collate statistics from large bodies of data, for complex tasks such as Enterprise Resource Planning (ERP), for mass-data manipulation such as Customer Relationship Management (CRM) applications, and for financial transaction processing.

Mainframe computers—*big iron* to some—often descend from the venerable IBM System/360 family, including modern mainframe machines such as IBM's zSeries, System z9, and System z10 servers. Other mainframe vendors include Unisys, Hewlett-Packard, Fujitsu, NEC, and so on. Your environment may feature a somewhat older—but still reliable—IBM AS/400 or other mainframe computer.

IBM estimates that 90% of their mainframes have Customer Information Control System (CICS) transaction processing software installed, and that their IMS and DB2 database software is also popular. Many IBM mainframes also run WebSphere MQ and WebSphere Application Server middleware—all enterprise-grade, mission-critical software that contains a great deal of data you'd probably like to use elsewhere.

Most mainframe applications store their data inside fairly standardized databases—DB2 being a star example on IBM mainframes. Accessing this data directly is straightforward, but ultimately not the best way to extend mainframe applications because the data itself doesn't contain any of the business logic. When we speak of extending a mainframe application, we usually want its business logic and other capabilities, not just the raw data. In fact, what would be ideal is to somehow transform the entire application from being a user interface (UI)-centric application into some kind of modern middle-tier component that encapsulates the application's business logic.

The Opportunities for Mainframe Applications

Let's take a simple example from my own experience: I used to work for a traditional, brick-and-mortar retailer of computer and video game software. Our business was run entirely on an IBM AS/400—in fact, our office didn't even include a Local Area Network (LAN) for several years because we all worked almost entirely on the AS/400 via 5250 terminal emulation. Even our email lived in the AS/400 (this was at the dawn of the public Internet, so email was strictly an internal thing at the time). Our AS/400 had a variety of prepackaged and custom-written (in the RPG programming language) applications that ran the business, from gathering sales transactions from our retail stores and processing restocking orders to communicating purchase orders to distributors and wholesalers to managing the product distribution center and inventory. The AS/400 even managed a fairly successful phone-in direct mail business, where customers would place orders by phone from paper catalogs, and we would fulfill those orders right from the main distribution center. A *lot* of business logic lived in the AS/400—how we handled inventory, how orders were processed, payment processing, and so on.

As the Internet gained in popularity and more of our customers got online, we wanted to offer an online shopping alternative. We were faced with a problem, though: All our data was wrapped up in the mainframe. It was *good* data, too—detailed product information, package sizes and weights (useful when calculating shipping), system compatibility information, and so on. All of our business processes were encapsulated in those applications, too. It wasn't enough to simply dump orders into a database; we wanted to use all the order-processing software we already owned—on the AS/400.

The AS/400 even had a robust customer order tracking system, which our mail order business unit relied upon and really liked. We *wanted* to simply re-use all that capability in a new, Web-based e-commerce application. Ideally, the online catalog would be built from the AS/400's database. Product availability would be taken from the AS/400's inventory information. That couldn't just come from database tables, though; *availability* in our world was a complex calculation involving on-hand stock, expected incoming shipments, existing commitments to our retail stores, and so forth. Ideally, orders would also be processed in the AS/400 and fulfilled just like our existing mail orders. We figured that we already had all the right pieces in place—we just needed to hook the Web site up to the AS/400. We didn't want to run the Web site *on* the AS/400—for a number of business reasons, we'd decided to use a farm of less-expensive Microsoft Windows-based Web servers—but we did figure that the entire backend could simply stay right where it was, on the '400.

We were wrong. We wound up implementing a complex system of data dumps and imports. We would dump data from the AS/400 each night, and import that data into a Microsoft SQL Server database that provided the backend for the Web site. Sales transactions were processed not by the AS/400 but by the individual Web servers using an all-new credit card processing infrastructure we had to build. Transactions were then exported from SQL Server and pulled into the AS/400 in a twice-daily import process so that order fulfillment could still take place within the mainframe—that import process had to leverage all the business logic in the AS/400 needed for order processing. All this importing and exporting still required hundreds of hours of custom RPG programming because we needed to make sure the incoming data was "clean" before dumping it into the AS/400's—because the other applications using that data assumed that anything in the database had been thoroughly validated. The entire project was painful, and it concerned us because we knew there would be other, similar data-sharing projects coming in the future. Looking back, I think that Web project may have been the beginning of the end for the AS/400 in our company. Great as it was at doing its job, we just couldn't keep investing in something that wouldn't expand and extend quickly when we needed it to.

So what was the problem? There were really three.

The Problem with Mainframe Applications

The big problem with mainframe applications is that they are generally monolithic. That is, the folks who wrote the application assumed that the application would be entirely self-contained, never need any major new capabilities, and never need any capabilities that the designers didn't think of in the first place. The application's presentation—generally screens designed for display on a terminal—business logic, data, and in many cases hardware are created in a single stack, often with very little differentiation between those layers. That makes it very difficult to break out *pieces* of the application, and while the application's data might be readily accessible, it's tough to find ways to leverage the application's business logic without extensive, expensive reprogramming.

The Hardware Problem

Some companies—especially those who are starting to eye their mainframe with a bit of distaste—focus on the mainframe hardware. It's expensive, although pricing has come down a bit as vendors begin to rely on commodity components such as Intel processors. Much of the hardware remains proprietary, and for companies who have moved beyond the need for the mainframe hardware, keeping it around—just because it happens to own all the important company data—can be painful.

For example, in the company I described earlier, we began moving more and more critical functions off the mainframe, utilizing an ever-more-complex arrangement of data transfers between the AS/400 and its PC-based successor systems. As more and more business processes began to depend on these delicate data dumps and imports, we began to see the AS/400 as a kind of weight around our necks. The hardware was expensive, the maintenance contracts were expensive, and we resented the mainframe simply because we felt it had taken our data hostage. We had a visceral feeling that simply ditching the hardware once and for all would solve a lot of our problems.

The Software Problem

The *real* problem, of course, lay mainly within the application software that ran on the mainframe. Mainframe applications are often written in what are powerful, but relatively primitive, programming languages like COBOL and RPG. These languages are well-suited to applications that need to process batches of data quickly, produce reports quickly, and interact with humans primarily through a text-based, field-oriented UI. These languages do *not,* however, lend themselves well to modern application development practices such as componentization, multi-tier development, and so forth.

Most modern languages—Microsoft's C#, Sun's Java, and even Web-oriented languages like PHP—are object-oriented and support object-oriented programming techniques such as encapsulation and inheritance. These, in turn, help foster a multi-tier development environment. For example, a modern application's UI is often little more than a set of basic data-entry routines. That data is submitted to a middle-tier component, whose job it is to validate the data before passing it to further tiers for processing. The business logic, in other words, resides in a discrete layer, enabling it to be used by not only the data-entry UI but by other means of input such as Web sites, automated business processes, and so forth.

Monolithic mainframe applications, however, tend to intermix the UI, business logic, and data management layers of an application. The *only* way to ensure that the right data gets into the application is to enter it into the UI. That's great for manual data entry but not so great when you begin looking for methods to leverage your data in other ways and to connect your data with other systems.

The Data Problem

Mainframe applications may use a wide range of database engines under the hood. Many are built on IBM's DB2 database; others may use internal, proprietary database engines. But even when the data is in a readily-accessible engine such as DB2, the data is often less than useful when it comes to connecting external applications.

One reason is the one I've already stated: You often can't just stick data into the database; you have to go through the application's entry screens to ensure the data is validated, that referential integrity is maintained, and so forth. My AS/400 experience included DB2, and we were delighted that Windows computers could communicate directly with the database using Open Database Connectivity (ODBC) drivers. We were dismayed when we realized that doing so would mean re-creating every scrap of business logic encompassed by our mainframe application so that the data we were sticking into DB2 would be valid and wouldn't cause problems for the many mainframe applications that utilized the data. In other words, just getting to the data didn't solve our problem: We basically needed to re-engineer all our application's business logic in a brand-new set of middle-tier components written in COM, .NET, or Java. That's a heavy task, even though we only needed to do it for *portions* of the mainframe application. We'd also have an ongoing maintenance problem, as any changes to business logic would then have to be reprogrammed in parallel—once on the mainframe, and once in the more modern middle-tier components.

What's the Solution?

Like anything in the IT world, there's never simply *one* solution. In the next part of this guide, I'll explore common solutions to the problem of using mainframe data in more modern scenarios. We'll look at the pros and cons of each solution, and try to come up with a wish list of capabilities and technologies that will lead to the best long-term solution.